

AD-A142 392

A PROBLEM EXPANDING PARAMETRIC PROGRAMMING METHOD FOR  
SOLVING THE JOB SHO. (U) CARNEGIE-MELLON UNIV  
PITTSBURGH PA MANAGEMENT SCIENCES RESEAR.  
G L THOMPSON ET AL. APR 84 MSRR-500

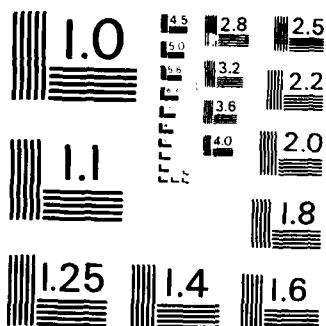
1/1

UNCLASSIFIED

F/G 5/1

NL





MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A

12

AD-A142 392

A PROBLEM EXPANDING PARAMETRIC  
PROGRAMMING METHOD FOR SOLVING THE  
JOB SHOP SCHEDULING PROBLEM

by

Gerald L. Thompson

and

Daniel J. Zawack

April 1984

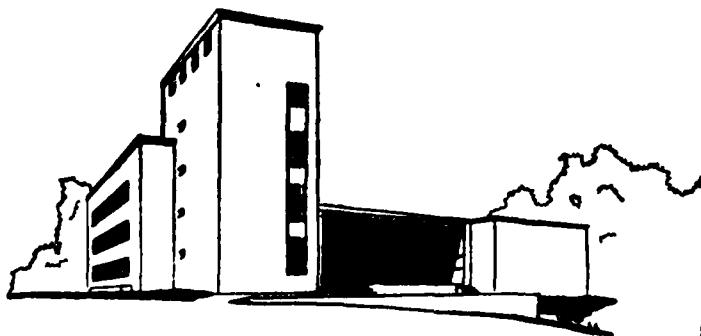
**Carnegie-Mellon University**

PITTSBURGH, PENNSYLVANIA 15213

**GRADUATE SCHOOL OF INDUSTRIAL ADMINISTRATION**

WILLIAM LARIMER MELLON, FOUNDER

DTIC FILE COPY



**DTIC**  
**ELECTE**  
JUN 26 1984  
**B**

**DISTRIBUTION STATEMENT A**

Approved for public release  
Distribution Unlimited

84 06 25 013

Management Science Research Report No. MSRR 500

A PROBLEM EXPANDING PARAMETRIC  
PROGRAMMING METHOD FOR SOLVING THE  
JOB SHOP SCHEDULING PROBLEM

by

Gerald L. Thompson

and

Daniel J. Zawack

April 1984

This report was prepared as part of the activities of the Management Science Research Group, Carnegie-Mellon University, under Contract No. N00014-82-K-0329 NR 047-048 with the U. S. Office of Naval Research. Reproduction in whole or in part is permitted for any purpose of the U.S. Government.

Management Sciences Research Group  
Graduate School of Industrial Administration  
Carnegie-Mellon University  
Pittsburgh, Pennsylvania 15213

**DISTRIBUTION STATEMENT A**

Approved for public release  
Distribution Unlimited

**DTIC**  
**ELECTE**  
JUN 26 1984  
**B**

A PROBLEM EXPANDING PARAMETRIC PROGRAMMING METHOD  
FOR SOLVING THE JOB SHOP SCHEDULING PROBLEM

by

Gerald L. Thompson and Daniel J. Zawack

ABSTRACT

A new zero one integer programming model for the job shop scheduling problem with minimum makespan criterion is presented. The algorithm consists of two parts: (a) a branch and bound parametric linear programming code for solving the job shop problem with fixed completion time; (b) a problem expanding algorithm for finding the optimal completion time.

Computational experience for problems having up to 36 operations is presented. The largest problem solved was limited by memory space, not computation time. Efforts are under way to improve the efficiency of the algorithm and to reduce its memory requirements.

KEY WORDS:

job shop scheduling algorithm  
branch and bound  
zero one integer program  
parametric linear programming  
problem expansion



Accession For	
MR	✓
DATE	
Volume	
PER FORM 50	
Distribution/	
Availability Codes	
Avail and/or	
Dist	Special
A-1	

## 1. INTRODUCTION

The literature on the job shop scheduling problem is extensive. Some well known books devoted entirely to scheduling problems are: Muth and Thompson [11], Conway, Maxwell, and Miller [6], and Baker [1]. More recently an easy to read introductory text by French [7] has appeared and an in depth survey of the mathematics of job shop scheduling is given by Bellman, Esogbue and Nabeshima [4].

There are three well-known mathematical programming models for solving the job shop scheduling problem under the minimize makespan criterion. The first to appear was by Bowman [5]. It was very cumbersome and no computational testing of it has ever appeared in the literature. The second model was given by Wagner [14]. Story and Wagner [11] provided the only computational testing of the Wagner model and concluded that it could not be relied upon to find optimal solutions. The third model was stated as a general mixed integer program by Manne [10] and this has become the most common notational representation of the problem. Of the three models, Manne's is the most economical in terms of number of variables and constraints. Unfortunately, its application has not yielded any significant computational success. Balas [3] has shown that the linear programming relaxation of this model is generally quite weak implying that often a large gap exists between the linear programming optimum and the integer programming optimum. Closing this gap is computationally very expensive. Balas [2] recognized that the Manne formulation is a disjunctive program and has applied disjunctive programming methods to strengthen the formulation [3], but no computational experience with this has been reported.

In this paper a new model for solving the job shop scheduling problem under the minimal makespan criterion is presented. It is a pure zero-one

mathematical program, in which a zero-one variable  $x(j,t)$  is set equal to 1 if operation  $j$  starts its uninterrupted processing period at time  $t$  and is zero for all other times. These variables are used to construct a model which admits only schedules whose completion times are no greater than some prespecified horizon time. The model is embedded in an algorithm whose approach is to start with a known schedule which completes by time  $T$ . Then an instance of the model is solved which admits only schedules completing by time  $T-1$ . Solving this model means finding a schedule completing by time  $T-1$  or proving that none exists. The model is solved by applying parametric linear programming techniques in the context of a branch and bound search. When it can be shown that no schedule exists for some completion time  $T-1$  then the previously found solution at  $T$  is optimal.

If a horizon shortening technique is to be efficient then it is important for the initial value of  $T$  to be a good upper bound on the optimal value. In order to obtain good upper bounds a problem expansion technique is utilized which yields a good upper and lower bound in each stage of the expansion process.

The problem is defined and notation given in Section 2. In Section 3 the model is given and in Section 4 a parametric linear programming procedure for its solution is discussed. In Section 5 the problem expanding technique is defined. The algorithm has been implemented and preliminary computational experience is presented in Section 6. Finally, in Section 7 conclusions as to the computational potential of this method are stated and areas for potential improvement of the method are noted.

## 2. THE JOB SHOP SCHEDULING PROBLEM

The general job shop problem can be stated as follows. There are a set of  $I$  items numbered from 1 to  $I$  that have to be processed on  $M$  machines

which are numbered from 1 to M. The production of item i requires a sequence of  $s_i$  operations. The operations are indexed according to the following scheme. Let  $n_0 = 0$ , and index the operations for item i consecutively from  $n_{i-1} + 1$  to  $n_i$  where  $n_i = n_{i-1} + s_i$ . For each item i let  $b_i = n_{i-1} + 1$ . Let the total number of operations be N so that  $n_I = N$ . There are precedence relationships between the operations on each item which are incorporated in the indexing scheme in the following way. If j and k are indices of operations on item i and if  $j < k$  then operation j must be performed prior to the start of operation k. Each operation j must be performed on a specified machine  $q(j)$ . Define the set of operations on machine m to be  $J(m) = \{j | q(j) = m\}$ . Finally each operation j requires time  $d(j)$  to complete. The objective of the formulation to be discussed is to find a processing order on each machine such that all operations are completed by time T.

The precedence relations between operations imply the existence of a lower bound on the start time of any operation. The early start time for operation j on item i can be expressed as  $E_j = \sum_{k=b_i}^{j-1} d(k)$ . By convention, if the sum is empty its value is zero. The precedence relations in combination with the required completion time T also establish an upper bound on the start time of any operation. The latest start time for operation j on item i can be expressed as  $L_j = T - \sum_{k=j}^{n_i} d(k)$ . Note that for each operation j on item i the difference  $L_j - E_j = T - \sum_{k=b_i}^{n_i} d(k)$  which is a constant.

In order to complete all operations by time T three conditions must be satisfied. First each operation  $j \in N$  must begin its uninterrupted processing period between  $E_j$  and  $L_j$ , inclusive. Second, operation j on item i cannot begin until all of its predecessor operations on item i are

completed. The third condition is that at most one operation can be performed by a machine during any time period.

### 3. THE MODEL

These conditions are stated notationally in an integer constraint set. The zero-one variables are  $X(j,t)$  where

$$X(j,t) = \begin{cases} 1 & \text{if operation } j \text{ begins at time } t \\ 0 & \text{otherwise.} \end{cases}$$

The constraints which require that operation  $j$  be performed are

$$(1) \quad \sum_{t=E_j}^{L_j} X(j,t) = 1 \text{ for } j = b_i, \dots, n_i, \quad i = 1, 2, \dots, I.$$

The second class of constraints requires that the precedence relations are satisfied for each related pair of operations. The constraints of this group take the following form

$$(2) \quad - \sum_{t=E_{j-1}}^k X(j-1,t) + \sum_{t=E_{j-1}+d(j)}^{k+d(j)} X(j,t) \leq 0 \text{ for } k = E_{j-1}, \dots, L_{j-1},$$

$$j = b_i + 1, \dots, n_i, \quad i = 1, \dots, I.$$

These constraints state that operation  $j$  cannot start until operation  $j-1$  has been completed. They also allow operation  $j$  the freedom to start in any feasible time period after operation  $j-1$  is complete.

The third group of constraints guarantees that during a time period  $p$  at most one operation is assigned to a machine. There will be a constraint in this group for each period  $p$  in which the machine  $m$  may be in operation. The potential periods in which a machine  $m$  may be operating are specified as

follows. The earliest time that machine  $m$  may begin operating is at the earliest start time of any operation which must be processed on machine  $m$ . Let the early start period of  $m$  be  $S_m$  where

$$S_m = \min_{j \in J(m)} E_j.$$

The last period in which machine  $m$  may be operating is the latest period during which any operation which must be processed on machine  $m$  may still be undergoing processing. Let the last operating period of machine  $m$  be  $F_m$  where

$$F_m = \max_{j \in J(m)} \{L_j + d(j) - 1\}.$$

The set of machine constraints is

$$(3) \quad \sum_{j \in J(m)} \sum_{t=1}^{d(j)} X(j, t+p-d(j)) \leq 1 \text{ for } p = S_m, \dots, F_m, m = 1, \dots, M.$$

The inner sum over the indices  $t = 1, \dots, d(j)$  implies that if operation  $j$  were to start being processed on machine  $m$  at some time in which its uninterrupted processing period would include period  $p$  then this operation will occupy machine  $m$  at that time. The outer sum over the index set  $j \in J(m)$  includes all operations  $j$  on machine  $m$  that might undergo processing in period  $p$ . Since the right hand side is one at most one operation of those potentially available can be in process on machine  $m$  in period  $p$ .

#### 4. PARAMETRIC LINEAR PROGRAMMING SOLUTION METHOD

Any zero one solution satisfying (1), (2), and (3) for an associated job shop scheduling problem provides a feasible schedule which completes by time  $T$ . In order to find such a feasible schedule it would be desirable to find an objective function which could be imposed on the problem

so that the solution of the resulting linear program would yield a feasible zero one solution. If an  $X$  vector were known which was a feasible zero one solution then it could be used to define an objective for which solving the constraint set as a linear program would yield the original  $X$  vector. Specifically, for each  $X(j,t)$  with value one in the known solution, set the corresponding objective function coefficients to one. Maximize this objective function subject to (1), (2), and (3), allowing  $X(j,t)$  to be continuous.

However, a feasible solution is not known for the above problem so such an objective function cannot be prespecified; instead we give a procedure which can learn the appropriate objective function via parametric linear programming techniques. The desired outcome of this learning process is either to find an integer solution defining a schedule completing by time  $T$  or to prove no such schedule exists.

The process begins by setting all objective function coefficients to zero and attempting to find a feasible solution to (1), (2), and (3) as a linear program. If this linear program is found to be infeasible then there is no schedule which completes by time  $T$ . If a feasible solution to the linear program is found then it is examined to determine if it is integer. If the solution is integer then no further work is required. If not, then the process of learning an appropriate objective function which will draw out an integer solution, if one exists, is carried out via a branch and bound search procedure. Structuring the learning process as a branch and bound search procedure makes certain that if an integer solution exists it will be found, but also if there is no feasible integer solution then that fact will be proved.

The search is carried out by "driving in" or "driving out" a variable at each stage to fix it at one or zero. Variables to be driven in or driven out are

those with fractional values in the current optimal solution to the linear program. The operation of driving in a fractional basic variable is done by setting its objective function coefficient to one and reoptimizing the linear program as a maximization problem. The operation of driving out a fractional basic variable is done by setting its objective function coefficient to minus one and reoptimizing. The variable driving in process is successful if the value of the reoptimized linear programming solution is one unit greater than the previous optimal value. The variable driving out process is successful if the value of the reoptimized linear programming solution is the same as that of the previous optimal solution. In the case of a variable driven in, the success criterion means that after reoptimization the variable has reached the value of one and the value of all other variables previously driven in or out remain unchanged. In the case of a variable driven out the success criterion means that after reoptimization the variable has reached the value zero and the values of any other variables previously driven in or out remain unchanged. A success in driving in or out means that the fixed variables may form a part of a feasible integer solution. Failure means that the set of fixed variables cannot form a feasible integer solution in the way they are currently fixed. This is evidenced by the fact that failure implies one of the variables to be fixed at zero or one remains fractional at the optimum even though it's fractional value adversely effects the optimal solution value.

In order to formalize the driving in and driving out of variables process into a branch and bound search, some notation is required. Let variable  $V$  be the current target value which the reoptimized linear program should obtain if the driving in or driving out operation is successful. Let  $K$  be the current number of variables driven in or out. Let  $LIST$  be an ordered list of variables

$X(j,t)$  where  $LIST(i)$  is the variable in position  $i$  of the list.

#### BRANCH AND BOUND SEARCH ALGORITHM

##### STEP 0 Initialization

Set  $V \leftarrow 0$ ,  $K \leftarrow 0$ ,  $LIST \leftarrow \emptyset$

Go to STEP 1

##### STEP 1 Test for Feasibility and Variable Selection

If the current solution to the linear program is all integer set  
 $FEASIBLE \leftarrow 'TRUE'$  and STOP. Else find the largest fractional variable  $X(j,t)$ .  
If there are ties break them by selecting the  $X(j,t)$  with the largest  $t$ . Set  
 $K \leftarrow K + 1$  and  $LIST(K)$  equal to the chosen variable. Go to STEP 2.

##### STEP 2 Driving in Variables

Set  $V \leftarrow V + 1$ . Set the objective function coefficient of the variable  
 $LIST(K)$  to plus one and reoptimize the resulting linear program. If the optimal  
solution value is equal to  $V$  go to STEP 1. Else go to STEP 3.

##### STEP 3 Driving out Variables

Set  $V \leftarrow V - 1$ . Set the objective function coefficient of the variable  
 $LIST(K)$  to minus one and reoptimize the resulting linear program. If the optimal  
solution value is equal to  $V$  go to STEP 1. Else go to STEP 4.

##### STEP 4 Drop Variable

Set the objective function coefficient of the variable  $LIST(K)$  to zero.  
Remove  $LIST(K)$  from  $LIST$  Set  $K \leftarrow K - 1$ . If  $K > 0$  go to STEP 5. Else set  
 $FEASIBLE \leftarrow 'FALSE'$  and STOP.

##### STEP 5 Backtracking

If the objective function coefficient of the variable  $LIST(K)$  is plus  
one, go to STEP 3. Else go to STEP 4.

END.

If the algorithm stops at STEP 1, then a feasible integer solution has been discovered which gives a schedule completing by time  $T$ . If it stops at STEP 4, then the problem has no solution completing by time  $T$ .

### 5. PROBLEM EXPANSION

Next this job shop scheduling model and the BRANCH AND BOUND SEARCH ALGORITHM are incorporated into an algorithm for finding an optimal makespan solution. The model and BRANCH AND BOUND SEARCH ALGORITHM provide a method of proving whether or not there exists a feasible solution completing by time  $T$ . Hence the optimal completion time may be determined by beginning with any feasible schedule and cutting off the horizon incrementally until an infeasible horizon is reached. If this horizon shortening procedure is to be efficient, a good starting upper bound is needed to reduce the number of time horizons examined.

In order to generate good upper bounds a problem expanding approach is utilized as follows. Initially, the job shop scheduling problem with only the first operation of each item is solved. This is a trivial problem since assigning the jobs on the required machines in any order is optimal. At stage  $z$  the problem  $P_z$  of optimally scheduling only the first  $z$  operations on each item is solved via horizon shortening. The optimal completion time for  $P_z$  is  $T_z$ . In order to find a good upper bound  $UB_z$  on problem  $P_z$  from which to begin the horizon shortening, the optimal schedule from problem  $P_{z-1}$  is used as a schedule for the first  $z-1$  operations on each item. Then all of the  $z$ th operations of the items are scheduled by sequentially appending them to the optimal schedule for  $P_{z-1}$  according to the rule that the operation with the earliest feasible start time is scheduled next. The variable  $UB_z$  is assigned the value of the resulting completion time of this heuristic schedule. The

horizon shortening begins by applying the model to seek a schedule completing by  $T = UB_z - 1$ . The optimal solution  $T_{z-1}$  to problem  $P_{z-1}$  provides a lower bound  $LB_z$  on problem  $P_z$ . Hence if a schedule is found for problem  $P_z$  which completes by  $LB_z = T_{z-1}$  then no further shortening is necessary. Let  $S = \max_{i=1, \dots, I} s_i$ . When subproblem  $P_S$  is solved, the optimal completion time for the entire problem is  $T_S$ .

In order to formalize the problem expanding method define  $P_z(T)$  to be the model of the job shop scheduling problem which includes only the first  $z$  operations of each item and admitting only schedules which complete by time  $T$ . Then the algorithm for finding an optimal makespan solution can be stated as follows:

PROBLEM EXPANDING ALGORITHM FOR JOB SHOP SCHEDULING.

STEP 1 Initialization

Set  $z \leftarrow 1$ . Solve  $P_z$ . Go to STEP 2.

STEP 2 Check for Inclusion of all Operations

If  $z = S$  STOP. The optimal completion time is  $T_S$ . Else set  $z \leftarrow z + 1$  and go to STEP 3.

STEP 3 Find Upper Bound and Lower Bound on Current Subproblem

Set  $LB_z \leftarrow T_{z-1}$  and determine the heuristic upper bound  $UB_z$ . Set  $T \leftarrow UB_z$  and go to STEP 4.

STEP 4 Determine whether there is a Schedule Completing by  $T-1$

Solve the model  $P_z(T-1)$  via BRANCH AND BOUND SEARCH ALGORITHM. If FEASIBLE = "FALSE" set  $T_z \leftarrow T$  and go to STEP 2. Else go to STEP 5.

STEP 5 Horizon Shortening

Set  $T \leftarrow T - 1$ . If  $T = LB_z$  set  $T_z \leftarrow T$  and go to STEP 2. Else go to STEP 4.

END.

In the problem expansion solution process for computation it is necessary to generate  $S-1$  linear programs in order to solve the  $S-1$  non-trivial subproblems. Furthermore the computational cost of finding an initial feasible solution to each linear program via a phase I procedure can be eliminated by the use of a crash start. The crash start for subproblem  $P_z$  is considered first. Then the method for solving  $P_z$  using a single linear programming formulation will be indicated.

Begin the horizon shortening procedure at stage  $z$  by setting up the model  $P_z(UB_z)$ . A feasible solution to  $P_z(UB_z)$  is defined by setting  $X(j,t)$  to one for  $t$  equal to the start time of operation  $j$  in the schedule defining the completion time  $UB_z$  and zero otherwise. Hence a feasible tableau for  $P_z(UB_z)$  can be obtained by pivoting into solution the variables with value one in the feasible solution corresponding to the schedule defining  $UB_z$ . This crash start can be obtained by  $N$  pivot steps.

It is necessary to set up only  $S-1$  linear programs, one for each non-trivial subproblem  $P_z$ , because in general the model  $P_z(T-1)$  can be obtained from the model  $P_z(T)$ . This can be accomplished in either of two ways. One way is to set the objective function coefficients of the variables  $X(n_i, t_i)$  for  $t_i = T - d(n_i)$ ,  $i = 1, \dots, I$  corresponding to operations completing at time  $T$  in model  $P_z(T)$ , to minus one and reoptimize via a primal simplex algorithm. Since the objective function is maximization these variables will only appear in a basic optimal solution if there is no feasible continuous solution which completes by time  $T-1$ . Such a situation will be evident if the optimal value of the objective function is negative. The second way to obtain model  $P_z(T-1)$  from  $P_z(T)$  is to add the constraint

$$\sum_{i=1, \dots, I} x(n_i, t_i) \leq 0$$

where again  $t_i = T - d(n_i)$ . Then reoptimize the current linear program via a dual simplex algorithm. Either of these methods of obtaining model  $P_2(T-1)$  from model  $P_2(T)$  means that upon obtaining  $P_2(T-1)$  one already knows whether or not the linear program is feasible and if it is feasible, already has a basic solution with which to begin BRANCH AND BOUND SEARCH ALGORITHM.

## 6. COMPUTATIONAL EXPERIENCE

The algorithm has been used to solve 44 test problems which had from four to six machines and from four to eight items. In the thirty five problems for which results are shown in Tables I - V, every item is processed exactly once or each machine. The processing order for each item was chosen by a random draw. Each operation required a unit time to complete. It should be noted that even though all operations have unit times the problem is NP hard as long as the number of machines is at least three, see [8]. Unit operation times were used to hold down the sizes of the linear programs to be solved.

The dimensions of the linear programs depend upon the horizon time  $T$  and the total processing time for each item. For item  $i$  the number of variables required is

$$(T - \sum_{j=b_i}^{n_i} d(j) + 1) s_i$$

corresponding to the number of possible starting times item  $i$  has. Note that for an item with a long total processing time relative to  $T$  only a few variables are needed, but for an item with a short total processing time relative to  $T$  many variables are needed. For item  $i$  there are  $s_i$  constraints of the form of (1) needed and there are

$$(T - \sum_{j=b_i}^{n_i} d(j) + 1) (s_i - 1)$$

constraints of the form of (2) needed. The number of constraints of the form of (3) depends on  $T$ , the operation time data and the precedence relations. An upper bound on the number of constraints of the form of (3) is  $T \cdot M$ . The memory available on the computer used for testing the algorithm limited the testing to linear programs having at most 350 variables and 400 constraints. The linear programs for the problems in Tables II, IV and V all usually approached these limits.

According to the algorithm, a series of models  $P_z(T)$  for  $z = 1, 2, \dots, S$  must be solved in order to solve an entire problem. Such a model is solved by either proving it is infeasible or else by learning a feasible integer solution for it. For the thirty five problems of Tables I - V, the most common way in which an infeasible model was shown to be infeasible was by proving that the linear programming relaxation corresponding to the model was infeasible. In the 149 subproblems which were proven to be infeasible in order to solve this group of 35 problems, there was only one subproblem for which the linear programming relaxation had a feasible solution while the corresponding integer program was infeasible. In this case the driving in and driving out operations failed for the first variable selected. This proved that while a feasible linear programming solution existed there was no feasible integer programming solution. This occurred in problem 4 of Table V.

For the problems in Tables I - V, the most common way of learning an integer solution, in a model which contained one, was by carrying out a sequence of driving in operations, without ever resorting to the driving out operation or variable dropping. Furthermore, usually fewer than  $\frac{N}{2}$  variables had to be

driven in before an integer solution emerged. Of all the models solved which had integer solutions for the problems in Tables I - V, only 6 models required the driving out operation and in only one model was it actually necessary to drop variables and backtrack. The problems in which these operations were required are indicated in the notes associated with each table.

The CPU times in Tables I - V show that when the number of items is equal to the number of machines the solution times are usually not long, but when the number of items exceeds the number of machines the solution time increased markedly. The long solution times are due to the fact that some of the linear programs require a large number of pivots to solve and the linear programming code which was implemented was not very sophisticated. The long solution times do not result from long branch and bound searches. The largest branch and bound search tree contained 16 nodes, but most had only 4 or 5 nodes.

In the nine problems for which results are shown in Table VI each item is processed on a given machine at most once, but each item may not need to be processed on every machine. The processing order for each item is again chosen at random. The operation times for these problems were chosen at random from the integers between one and nine. The first seven of these problems were generated by the authors' code. The eighth problem is taken from Lenstra [9] and the ninth problem is taken from Nemeti [12].

For the problems of Table VI every model instance which was infeasible was proven to be infeasible by showing that the linear programming relaxation of the model was infeasible. Also for every model instance which had a feasible integer solution, an integer solution was learned by carrying out a sequence of driving in operations, without ever resorting to the driving out operation or variable dropping. Again, usually fewer than  $\frac{N}{2}$  variables had to be driven in before an integer solution was learned. As with the unit time

problem the long solution times result from the inefficiency of the linear programming code and not from long branch and bound searches.

## 7. CONCLUSIONS

The computational experience presented supports three observations which provide encouragement in regard to the computational potential of this approach. First, integer infeasibility of a constraint set defined by (1), (2), and (3) is usually equivalent to infeasibility in the linear programming relaxation of that constraint set. Second, when integer solutions to a constraint set defined by (1), (2), and (3) exist, they frequently are learned by carrying out a sequence of consecutive driving in and/or driving out operations on variables. The operation of dropping a selected variable and backtracking is seldom required. Third, usually fewer than  $\frac{N}{2}$  driving in or driving out operations need to be carried out in order to learn an integer solution. Therefore while the branch and bound search algorithm is, in the worst case, a complete enumeration method, preliminary computational experience above indicates that for problems of the sizes considered, the search trees are quite small and hence are not computationally difficult for the method. In other words, the method given in this paper has been limited to solving relatively small problems because of the memory requirements of the resulting linear programming problems that have to be solved and not by the computational effort.

In order to determine whether these results will hold up for larger problems, future work will focus on reducing memory requirements and improving the efficiency of the linear programming solution method. Work on these problems is contemplated from two directions. The first is to study the possibility of reducing the size of the problem by preprocessing. The number of variables can be reduced by  $N$  if each of the equations of the form of (1) is solved and substitution is made. A search for redundant constraints may also be implemented.

The second direction in which improvement will be sought is via the implementation of the Pivot and Probe Algorithm [13]. This is an algorithm which attempts to take advantage of redundancy in the constraint set to improve efficiency by updating only a relevant subset of the constraints when carrying out a pivot. The Pivot and Probe Algorithm also reduces memory requirements. This algorithm has yielded 80 percent reduction in computation time required for randomly generated linear programming problems relative to codes which update the full tableau.

If these efforts are able to increase the efficiency of the linear program solution process and to reduce memory requirements for the algorithms in this paper then it will be possible to test them on larger problems.

TABLE I

NOTES

M = 4, I = 5, N = 20

The processing order was chosen at random and all operation times are one. Problem 3 required the dropping of 4 variables and 4 backtrack steps. No other problem required variables to be dropped or driven out

PROBLEM	CPU TIME IN SECONDS
---------	------------------------

1	6
2	4
3	21
4	3
5	4

Avg. 7.6

TABLE II

NOTES

M = 4, I = 8, N = 32

The processing order was chosen at random and all operation times are one. None of the problems required the driving out operation or dropping of variables and backtracking.

PROBLEM	CPU TIME IN SECONDS
---------	------------------------

1	83
2	141
3	840
4	171
5	285

Avg. 304

TABLE III

NOTES

M = 5, I = 5, N = 25

The processing order was chosen at random and all operation times are one. Problem 1 required the driving out operation on one variable and Problem 7 required it on two variables. The remaining problems did not require variable dropping or driving out.

PROBLEM CPU TIME  
IN SECONDS

1	32
2	2
3	10
4	9
5	5
6	29
7	20
8	6
9	16
10	16

Avg. 14.5

TABLE IV

NOTES

M = 5, F = 7, N = 35

The processing order was chosen at random and all operations times are one. Problems 2, 4, and 5 each required the driving out operations on one variable. The other problems did not require variable dropping or driving out.

PROBLEM CPU TIME  
IN SECONDS

1	416
2	171
3	843
4	1361
5	261

Av.g 370.4

TABLE V

NOTES

M = 6, I = 6, N = 36

The processing order was chosen at random and all operation times are one. In problem 4 a linear program in  $P_6$  was feasible while the corresponding integer program was feasible. The driving out operation was required for one variable in problem 9. No other problem required variable dropping or driving out.

PROBLEM	CPU TIME IN SECONDS
1	67
2	175
3	116
4	168
5	137
6	132
7	114
8	3
9	395
10	38
Avg.	134.5

TABLE VI

NOTES

M = 4

Processing orders for each item were chosen at random. The processing times vary between 1 and 9. Neither the driving out operation nor dropping of variables was required for any of these problems. Problem 8 is taken from [8] and problem 9 is taken from [12].

PROBLEM	ITEMS	OPERATIONS	CPU TIMES IN SECONDS
1	5	18	438
2	5	18	196
3	5	18	8
4	5	18	174
5	6	22	57
6	6	24	1212
7	6	24	360
8	4	13	1297
9	5	13	34
Avg.			386.2

## 8. REFERENCES

- [ 1] Kenneth R. Baker, Introduction to Sequencing and Scheduling, John Wiley and Sons, New York, 1974.
- [ 2] E. Balas, "Machine Sequencing Via Disjunction Graphs: An Implicit Enumeration Algorithm," Operations Research 17: 941-957, 1969.
- [ 3] E. Balas, "Disjunctive Programming and a Hierarchy of Relaxations for Discrete Optimization Problems," Management Science Research Report MSRR 492, Carnegie-Mellon University.
- [ 4] R. Bellman, A. E. Esogbue and I. Nabeshima, "International Series in Modern Applied Mathematics and Computer Science," Volume 4: Mathematical Aspects of Scheduling and Applications, Pergamon Press, Oxford, 1982.
- [ 5] E. H. Bowman, "The Scheduling Sequencing Problem," Operations Research 7(5): 621-624, 1959.
- [ 6] R. W. Conway, W. L. Maxwell, and L. W. Miller, Theory of Scheduling, Addison Wesley, Reading Mass. 1967.
- [ 7] Simon French, Sequencing and Scheduling: An Introduction to the Mathematics of the Job Shop, Ellis Horwood Ltd., Chichester, England, 1982.
- [ 8] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP Completeness, Freeman, 1979.
- [ 9] J. K. Lenstra, Sequencing by Enumeration Methods, Mathematical Center Tract 67 Mathematisch Centrum, Amsterdam, 1974.
- [10] A. S. Manne, "On the Job Shop Scheduling Problem," Operations Research 8(2): 219-223, 1969.
- [11] John F. Muth and Gerald L. Thompson (editors), Industrial Scheduling. Prentice Hall Inc. Englewood Cliffs, New Jersey, 1963.
- [12] L. Nemeti, "Das Reichenfolge problem in der Fertigungsprogrammierung und Linearplanung mit logischen Bedingungen," Mathematika 6(29), 1964, 87-99.
- [13] A. P. Sethi and G. L. Thompson, "The Pivot and Probe Algorithm for Solving a Linear Program," Mathematical Programming, April 1984.
- [14] A. E. Story and H. M. Wagner, "Computational Experience with Integer Programming for Jobshop Scheduling," in [11], pp. 207-219.
- [15] H. M. Wagner, "An Integer Linear Programming Model for Machine Scheduling," Naval Research Logistics Quarterly 6(2), 131-140, 1959.

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER MSRR #500	2. GOVT ACCESSION NO. DA-1172-312	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Problem Expanding Parametric Programming Method for Solving the Job Shop Scheduling Problem	5. TYPE OF REPORT & PERIOD COVERED Technical Report April 1984	
7. AUTHOR(s) Gerald L. Thompson Daniel J. Zawack	6. PERFORMING ORG. REPORT NUMBER MSRR 500	
PERFORMING ORGANIZATION NAME AND ADDRESS Graduate School of Industrial Administration Carnegie-Mellon University Pittsburgh, Pennsylvania 15213	8. CONTRACT OR GRANT NUMBER(s) N00014-82-K-0329 NR 047-048	
9. CONTROLLING OFFICE NAME AND ADDRESS Personnel and Training Research Programs Office of Naval Research (Code 434) Arlington, VA 22217	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	12. REPORT DATE April 1984	
	13. NUMBER OF PAGES 20	
	15. SECURITY CLASS. (of this report)	
16. DECLASSIFICATION/DOWNGRADING SCHEDULE		
11. DISTRIBUTION STATEMENT (of this Report)		
<div style="border: 1px solid black; padding: 5px; text-align: center;"> <b>DISTRIBUTION STATEMENT A</b>                      Approved for public release                      Distribution Unlimited                 </div>		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)  job shop scheduling algorithm, branch and bound, zero one integer program parametric linear programming, problem expansion		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) A new zero one integer programming model for the job shop scheduling problem with minimum makespan criterion is presented. The algorithm consists of two parts: (a) a branch and bound parametric linear programming code for solving the job shop problem with fixed completion time; (b) a problem expanding algorithm for finding the optimal completion time. Computational experience for problems having up to 30 operations is presented. The largest problem solved was limited by memory space, not computation time. Efforts are under way to improve the efficiency of the algorithm and to reduce its memory requirements.		

DD

FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ENCLOSURE

FILED